Universidad Complutense de Madrid

Facultad de Informática

Desarrollo de HFOSS para ayudar en desastres naturales, como parte del proyecto Sahana

Autor: **Jesus Iráizoz Domínguez**
Directora: **Sara Román Navarro**

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Curso 2017/2018

Madrid, junio de 2018

Agradecimientos

Antes de comenzar, me gustaría agradecer a aquellos que, de algún modo, han contribuido a la realización de este trabajo:

- A mi directora del proyecto, Sara Román Navarro, cuya ayuda, interés y confianza depositada, así como el altruismo que me ha hecho ver en la informática, han sido claves para superar y realizar este trabajo.
- A mis compañeros de las prácticas en Beyond Soluciones y Sistemas, por la confianza y ánimo que me han dado y por todo lo que me han enseñado.
- A mis familiares y amigos, que me han soportado, me han ayudado y me han inspirado desde el primer día hasta el último.
- Por último, a mis padres y mi hermana, porque han sido mis mentores, profesores, psicólogos, compañeros, ..., mi vida desde el día en que nací.

«Vivimos en el mundo cuando amamos, sólo una vida vivida para los demás merece la pena ser vivida.» (A. Einstein)

Resumen

Cualquier persona del mundo, independientemente de dónde resida, se encuentra en riesgo de verse afectada por cualquier tipo de desastre, bien sea natural o humano. Por ello, los Estados, disponen de diferentes mecanismos y organizaciones para la gestión de los diferentes tipos de desastres, como las organizaciones de Protección Civil de cada país. No obstante, existen multitud de fundaciones y organizaciones no gubernamentales (ONG) que dedican sus esfuerzos a realizar estas mismas tareas, ya sea dando apoyo a las agrupaciones estatales o sustituyéndolas allí donde no existen.

Ante una situación de emergencia, es fundamental mantener una buena organización y coordinación, con el ánimo de poder responder de manera rápida y efectiva. Es, por tanto, necesaria la utilización de herramientas que ayuden a ello, y más aún en aquellas situaciones en las que el caos es el principal actor.

Ofrecer una herramienta que dé soporte a estas situaciones es la meta que persigue este proyecto, especializándose en la gestión de las incidencias y los vehículos de emergencia.

Palabras clave

- Sahana
- HFOSS
- Desastres
- Vehículos de emergencia
- Localización
- Software libre
- .NET Core
- ASP
- Android

Abstract

Any person in the world, no matter where he resides, is at risk of being affected by any type of disaster, whether natural or human. For this reason, the States have different mechanisms and organizations for the management of disasters, such as Civil protection organizations of each country. However, there are many foundations and non-governmental organizations (NGOs) that dedicate their efforts to perform these same tasks, either by giving support to state groups or replace them where they do not exist.

In the face of an emergency, it is essential to maintain a good organization and coordination, to respond in a quick and effective way. It is therefore necessary to use tools that help them, and even more so in those situations in which chaos is the main actor.

The aim of this project is to offer a tool that supports these situations, specializing in the management of incidents and emergency vehicles.

Key words

- Sahana
- HFOOS
- Disasters
- Emergency vehicles
- Tracking
- Open-Source
- .NET Core
- ASP
- Android

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

A lo largo de este documento, se pretende explicar el proyecto desarrollado como Trabajo Final de Grado durante el último curso del grado en Ingeniería Informática, cursado en la Universidad Complutense de Madrid.

La necesidad de este proyecto, nace desde la Fundación de Software Sahana, la cual, se dedica a desarrollar código abierto con fines humanitarios (HFOSS, *Humanitarian Free Open-Source Software*).[1]

Al ser una comunidad de código abierto, está abierta a que cualquiera pueda colaborar, bien sea desarrollando o probando y reportando errores. En el caso del desarrollo, nos presentan una serie de *blueprints* a desarrollar, bien sea del *framework* o módulos.

En nuestro caso, se pretende desarrollar un sistema capaz de administrar el parque móvil de una organización, controlar la posición de los vehículos y gestionar las incidencias que existan en una zona o territorio.

Para ello, se creará una aplicación web y un prototipo de aplicación móvil con comunicación entre ambas por medio de una API.

1.1. Antecedentes

De entre las múltiples opciones que se ofrecen desde Sahana, para este proyecto, se ha elegido un módulo para la gestión de vehículos de emergencia.

Este módulo, únicamente dispone de una escueta descripción, con pocos detalles técnicos y especificaciones y sin ningún código con el que continuar el desarrollo. [2]

Por tanto, hemos tenido la obligación de definir un proyecto desde el inicio, realizando una captura de requisitos, dando las especificaciones del sistema y desarrollándolo e implementándolo por completo.

[1] Para más información, puede visitarse la página web de la fundación en: https://sahanafoundation.org/

[2] Proyecto anterior: *Vehicle Management*

Para este trabajo, hemos tenido que realizar cuatro tareas fundamentales para el desarrollo del sistema:

1. Configuración del servidor y la base de datos
2. Desarrollo de la aplicación web
3. Desarrollo de un prototipo de aplicación móvil
4. Comunicación de la aplicación web con la aplicación móvil

1.2. Objetivos

Este trabajo de fin de grado, tiene como objetivo principal desarrollar un sistema capaz de dar soporte en la gestión de situaciones de emergencia, en concreto, en lo relacionado con los vehículos de emergencia.

A su vez, este objetivo principal, se divide en otros secundarios, que son:

- Gestionar el parque móvil, las incidencias de una zona geográfica y conocer el estado y posición de los vehículos.
- Conseguir que las organizaciones/clientes que utilicen este sistema sean capaces de responder de manera rápida a las situaciones de emergencia que surjan, con el fin de dar un buen servicio y minimizar el tiempo de respuesta.
- Dar un proyecto estructurado y con suficiente información como para poder seguir desarrollándolo

1.3. Estructura del documento

De aquí en adelante, la estructura utilizada para este documento es la siguiente:

- **Capítulo 3 Estado del arte**: se muestra el proyecto original en el que se basa este TFG y se realiza un repaso por las alternativas existentes en el mercado. Del mismo modo, se definen una serie de ideas claves a tener en cuenta para el desarrollo del sistema.
- **Capítulo 4 Descripción general**: se presentan las especificaciones del sistema, con las tecnologías y herramientas utilizadas, así como las características *hardware* a utilizar.
- **Capítulo 5 Planificación**: indica la planificación que se ha seguido para la realización del trabajo y describe las tareas desempeñadas, teniendo en cuenta las cuatro tareas necesarias para el desarrollo.
- **Capítulo 6 Resultados y Trabajo Futuro**: se realiza una muestra del trabajo desarrollado, las especificaciones que se han logrado y se plantean las líneas a seguir para completar el trabajo

- **Capítulos 7 y 8 Conclusiones**: para finalizar, se exponen las conclusiones del autor en relación con lo vivido y aprendido.

Chapter 2

Introduction

In this document, we pretend to explain the project developed as Final Degree Project during the last year at Computer Science, in Complutense University of Madrid.

The need of this project comes from Sahana Software Foundation, which is focus on the development of Humanitarian Free Open-Source Software (HFOSS).[1]

As it's an open source community, it's open for everybody to collaborate, developing or testing and reporting errors. In developing, you can find a list of blueprints for develop the framework o modules.

In our case, we pretend to develop a system, able to manage a fleet, track vehicles and manage incidences over an area. For all that, we'll develop a web application and a mobile prototype, with an API for intercommunication.

2.1. Precedent

From the multiple options offered by Sahana for this project, we've choosed a fleet emergency manager module.

This module just have a description, without entering into technical details, neither including any source code for its development. [2]

Therefore, we indebted to define from the beginning a software project, getting the requirements, defining the system specifications and developing and implementing it completely.

For this work, we had to carry out four fundamental tasks for the system development:

1. Server and data base configuration
2. Web application development
3. Mobile application development
4. Applications comunication

[1] For more info, you can visit their web site: https://sahanafoundation.org/

[2] Previous project:Vehicle Management

2.2. Goals

The main goal of this project is to develop a system to support emergency situations, concretely those related to emergency vehicles.

As secondary objectives, we have:

- Manage a vehicle fleet and incidents over an area and vehicle tracking.
- Achieve client organizations to be able to dispatch quickly emergency situations, for offer better service and minimize response time
- Offer an organized project with enough information to continue its development

2.3. Document structure

From here on out, we'll use the following structure:

- **Chapter 3 State of the Art**: we will review the original project in which this project is based and the market existent alternatives. Also, a series of ideas will be defined for understand the development.
- **Chapter 4 General Description**: the system specifications, with the tools and technologies used, as well as the hardware specifications used.
- **Chapter 5 Planning**: the planning followed to carry out this work, describing the executed tasks.
- **Chapter 6 Results and Future Work**: fulfilled work and reached specifications are shown as well as lines of action for its future development
- **Chapter 7 & 8 Conclusions**: author's conclusions about all learned during this project.

Chapter 3

Estado del arte

En este capítulo, se pretende ver el proyecto primitivo, que ha servido como base para el desarrollo de este; y las alternativas existentes en el mercado.

3.1. Proyecto primitivo

Como se ha comentado en el capítulo anterior, este trabajo está basado en otro proyecto anterior del cual se disponía una breve descripción.

Se trata de un módulo diseñado para la aplicación Sahana EDEN, en el que se presenta una aplicación multiplataforma, capaz de convivir con otros sistemas con el fin de mejorar sus características.

En la descripción, se definía que el sistema debía ser capaz de:

- Desplegar una unidad de emergencia.
- Localizar un vehículo
- Puntuar a los conductores
- Gestionar el mantenimiento de los vehículos

El sistema, ofrecía una aplicación como conjunto de paquetes que, de manera individual, alcanzaban las tareas anteriormente descritas, y en conjunto formaban el objetivo principal.

No obstante, como ya se ha comentado, tan solo es un proyecto (o idea) para realizar, ya que realmente, no se contaba con ningún tipo de documentación técnica ni código fuente desarrollado.

A continuación, se presentan las funcionalidades que, según este proyecto, la aplicación debería tener. Estas funcionalidades, servirán como base para el diseño del sistema que hemos desarrollado.

3.1.1. Despliegue

Una de las principales funciones era la del despliegue de vehículos. Pensada para aquellos que necesiten ayuda, la aplicación sería capaz de comunicarlos con las unidades desplegadas de manera fiable y rápida.

Figure 3.1: Diseño del sistema anterior

Proponían la idea de la creación de un centro de llamadas, con varios operadores que se encargasen de recibir las llamadas de emergencia. Además, proponen que el contacto, también pueda realizarse tanto por la página web como a través de la aplicación móvil.

También abogaban por una aplicación libre de fallos, ofreciendo un espacio donde informar sobre fallos y posibles mejoras para la aplicación, permitiendo de esta manera, ampliar las funcionalidades de la aplicación.

Con el software adecuado, el centro de despliegue sería capaz de adaptarse al cambio del tráfico y las condiciones meteorológicas. Sería capaz de modificar el mapa para ajustarse a las condiciones de las carreteras, las climatologías adversas y los informes de accidentes. Además, podría utilizarse para determinar el mejor vehículo disponible, bajo esas condiciones, para alcanzar el destino.

Por último, la aplicación permitiría mantener un inventario de vehículos organizado, de tal manera que, el sistema, utilizaría la información que ha ido aprendiendo acerca de las situaciones el tiempo y el tráfico para determinar el mejor vehículo para el trabajo a desempeñar.

3.1.2. Localización

La localización era una parte fundamental para esta aplicación. Se propone la utilización de un sistema de localización en tiempo real por GPS.

En concreto, integrando el software de localización Open GTS, incluyendo en cada vehículo un dispositivo de GPS para conocer la localización en cada momento.

Además, el software pretendía ofrecer un soporte en la creación de la ruta hasta la incidencia, en función de la posición de los vehículos y de las incidencias existentes, con el fin de ofrecer una respuesta más rápida.

3.1.3. Feedback

Con el fin de obtener una retroalimentación acerca de su aplicación, se habilitaría un cuestionario para informarse sobre los pros y contras de la aplicación.

Esto, permitiría al usuario dar nuevas ideas para mejorar el software, además de servir como contacto para informar sobre fallos.

3.1.4. Mantenimiento de vehículos

Para asegurar que cada vehículo se encontrase en las mejores condiciones, se planteó implementar una serie de métodos para planificar el mantenimiento de los vehículos.

Introduciendo su número VIN (*Vehicle Identification Number*, similar al número de bastidor), el usuario podría ver todos los documentos del vehículo así como su próxima revisión y puesta a punto planificada.

Si el vehículo, no cumpliese unas condiciones mínimas para ser utilizado, el sistema deshabilitaría el vehículo a la espera de recibir mantenimiento.

3.2. Alternativas del mercado

En el mundo de los servicios de emergencia, la función de control de despliegue, siempre ha estado muy presente desde que se instauraron los servicios de ambulancias con el fin de coordinar todas las llamadas de socorro.

A partir de la década de 1950, con la aparición de la radio, comenzó a comercializarse la idea de un punto único de respuesta a emergencias, que no se haría realidad hasta la aparición del 911 en EE.UU.

Hoy en día, los servicios de emergencia, al igual que otros servicios como el de taxi, cuentan con complejos y potentes sistemas CAD (*Computer-aided dispatch*, envío asistido por ordenador) que les permiten no sólo gestionar los vehículos y enviar unidades, sino también recibir las llamadas.

3.2.1. Security Patrol Software - Security Computer-Aided Dispatch Software

Esta aplicación, desarrollada por la empresa Kerkton Security Technologies LLC, ofrece una aplicación sencilla, fiable y potente basada en la nube.

Entre sus características, destacan:

- Envío asistido por ordenador (CAD)
- Gestión de fichas de personas
- Informes de incidentes
- Informes de actividad diarios
- Mensajería instantánea
- Localización en tiempo real

3.2.2. Intergraph Computer-Aided Dispatch (I/CAD)

La empresa Hexagon, ofrece una suite de gestión de incidentes, que agrupa diversas aplicaciones para gestionar llamadas, realizar despliegues, integrar los servicios en los vehículos y realizar análisis de seguridad.

En lo que nos atañe, la empresa ofrece un producto, I/CAD Call Handling & Dispatching, centrado en mejorar la velocidad y eficiencia de la recepción de llamadas y envío de unidades.

Entre sus capacidades, este producto ofrece:

- Seguimiento de llamadas, envíos y gestión de recursos
- Integración de voz, texto e información multimedia
- Información en tiempo real
- Información de cámaras de vídeo (CCTV, de tráfico, etc.)

Esta herramienta es utilizada por múltiples organizaciones de todos el mundo, como la policía Bávara, el cuerpo de Marines de los Estados Unidos o el metro de Louisville.

3.3. Análisis del sistema anterior y las alternativas

Con el sistema primitivo analizado, y habiendo estudiado las alternativas existentes en el mercado, se concluyeron una serie de ideas y características que debían tenerse en cuenta en el sistema:

- No existía ningún tipo de documentación o especificaciones sobre la aplicación original, exceptuando la breve descripción del proyecto, por tanto, debían especificarse qué requisitos y funcionalidades debería tener el sistema y cuáles no.
- Tampoco se encontró ningún código fuente en el que basarse o continuar desarrollando, por ello, también debía realizarse un diseño del sistema.
- Las alternativas anteriormente expuestas, son sistemas muy potentes, complejos y de pago, y no se encontró ningún sistema similar de *software* libre y gratuito, por lo que no existe una alternativa para organizaciones con escasos recursos, existiendo la oportunidad para crear un sistema similar a los CAD expuestos, pero más sencillo y manejable.

Por consiguiente, se vio la necesidad de realizar un proyecto de software completo, realizando una captura de requisitos (teniendo en cuenta la descripción del proyecto primitivo y las características de las alternativas del mercado), definiendo las funcionalidades, diseñando el sistema y desarrollándolo e implementándolo.

Chapter 4

Descripción General

El sistema, denominado GEPAME (Gestor de Parque Móvil de Emergencias), pretende, por un lado, ofrecer una solución libre y gratuita a organizaciones y entidades que estén encargadas de la gestión de vehículos de emergencia, y por otro, ofrecer un proyecto que sirva como nexo de unión entre el mundo real y el mundo universitario. Por ello, a lo largo de este capítulo, se van a detallar las diferentes tecnologías, hardware utilizado y el funcionamiento del sistema.

4.1. Sistema

GEPAME, es resultado de un proyecto de software iniciado desde cero, basado en la descripción de un módulo de Sahana EDEN y en la investigación de las características y funcionalidades de los sitemas CAD existentes en el mercado, el cual contempla las fases de desarrollo, comunicación e implementación.

Se trata de un sistema que consta de dos aplicaciones: una web, para la gestión y administración general, y otra móvil, diseñada para ser utilizada en los vehículos. Estas, se comunican entre si por medio de una API REST disponible desde la aplicación web.

4.1.1. Funciones del sistema

La aplicación web del sistema, debe ser capaz de realizar las siguientes funciones:

- Llevar un registro (inventario) de todos los medios de transporte (vehículos) de los que se disponga
- Controlar el estado de dichos vehículos, para poder programar su mantenimiento
- Almacenar una hoja de servicio (historial) de cada vehículo
- Controlar las entradas y salidas de los vehículos
- Crear, mostrar y gestionar las incidencias
- Controlar los usuarios de la aplicación y su nivel de acceso
- Conocer la posición de los vehículos desplegados

- Gestionar informes
- Asignar incidencias a vehículos
- Comunicase con otros sistemas GEPAME
- Interactuar con Sahana EDEN
- Ser capaz de trabajar sin Internet, a través de otros métodos de comunicación

Por su parte, la aplicación móvil debe contar con las funcionalidades:

- Mostrar y crear incidencias
- Mostrar la información de un vehículo
- Enviar en tiempo real la posición de un vehículo
- Crear y enviar informes
- Funcionar sin conexión a Internet, a través de otros métodos de comunicación

Por contra, se decidió que el sistema no contase con las siguientes características:

- Aplicación web
  - Gestión de RRHH (conductores)
  - Gestión de llamadas telefónicas
  - *Feedback*
- Aplicación móvil
  - Comunicación con otros sistemas
  - Gestión del parque móvil
  - Asignación de incidencias

4.1.2. Características de los usuarios

Para este sistema, se identifican tres tipos de usuarios distintos:

Usuario: el más básico, el que se encuentra en el vehículo. Debe tener los suficientes reflejos para visualizar una incidencia y la capacidad para decidir su importancia.

Gestor: este usuario dispone de privilegios medios y está orientado a las personas que se encarguen día a día de las tareas de gestión de incidencias. Deberá tener la suficiente capacidad para analizar las incidencias y responder en consecuencia, además del conocimiento de la aplicación.

Administrador: es el usuario con mayores permisos en el sistema, tiene control total sobre la aplicación. Debe ser una persona (o varias) con cierta responsabilidad y conocimiento de la aplicación, con conocimientos medios-altos de informática, para poder realizar la configuración tanto de la aplicación como del servidor en el que se desplegará.

4.1.3. Restricciones

A continuación, se establecen las restricciones de desarrollo necesarias para el sistema.

Limitaciones de hardware Dada la naturaleza del proyecto, en la que las organizaciones usuarias podrían no contar con suficientes recursos o economía, el sistema deberá ser lo suficientemente ligero y estar suficientemente optimizado como para ser ejecutado en un equipo sencillo, tanto a nivel económico como de recursos.

Lenguajes de programación El sistema web, deberá desarrollarse bajo el framework de Microsoft .NET Core y con una versión mínima v2.0. La aplicación móvil, puede desarrollarse tanto con código nativo como con otras plataformas que permitan el desarrollo multiplataforma.

Protocolos de comunicación El sistema dispondrá de una API tipo REST para la comunicación de la aplicación web con todas las aplicaciones móviles activas, así como con otros posibles sistemas que se integren posteriormente.

Cada aplicación móvil, deberá comunicarse única y exclusivamente con un sistema GEPAME al que se tenga acceso.

Adicionalmente, el sistema deberá ser capaz de comunicarse tanto por medio de Internet como mediante otras tecnologías que no utilicen Internet, sin interferir la una con la otra y con otros canales de comunicación.

Persistencia de datos GEPAME, en su aplicación web, almacenará la información en una base de datos relacional, y deberá ser capaz de acceder a ella independientemente del sistema gestor.

Seguridad La seguridad de los datos estará establecida por el sistema gestor de la base de datos. Todos los datos almacenados en la base datos cumplirán con la normativa vigente de protección de datos.

4.1.4. Interacción con otros sistemas

El sistema GEPAME, es un sistema autónomo e independiente de otros. No obstante, se pretende que GEPAME pueda interactuar con el sistema de Sahana Eden, para ampliar sus características y mejorar en la gestión de recursos; y con otros sistemas GEPAME, con el fin de ampliar su cobertura y la coordinación con otras entidades que lo utilicen.

La interacción con dichos sistemas, se realizará por medio de servicios web, preferiblemente mediante una API REST. En el caso de realizarse por medio de REST, se proporcionará un listado de métodos a los que llamar, además de un archivo de configuración Swagger en JSON, de tal manera que, los sistemas conectados, puedan obtener y enviar la información necesaria.

El sistema, funcionará principalmente a través de Internet, no obstante, puede darse el caso de que, en una situación de emergencia, la infraestructura de red (MAN o WAN) que ofrece dicho servicio se encuentre saturada o colapsada, por tanto, el sistema no podría comunicarse.

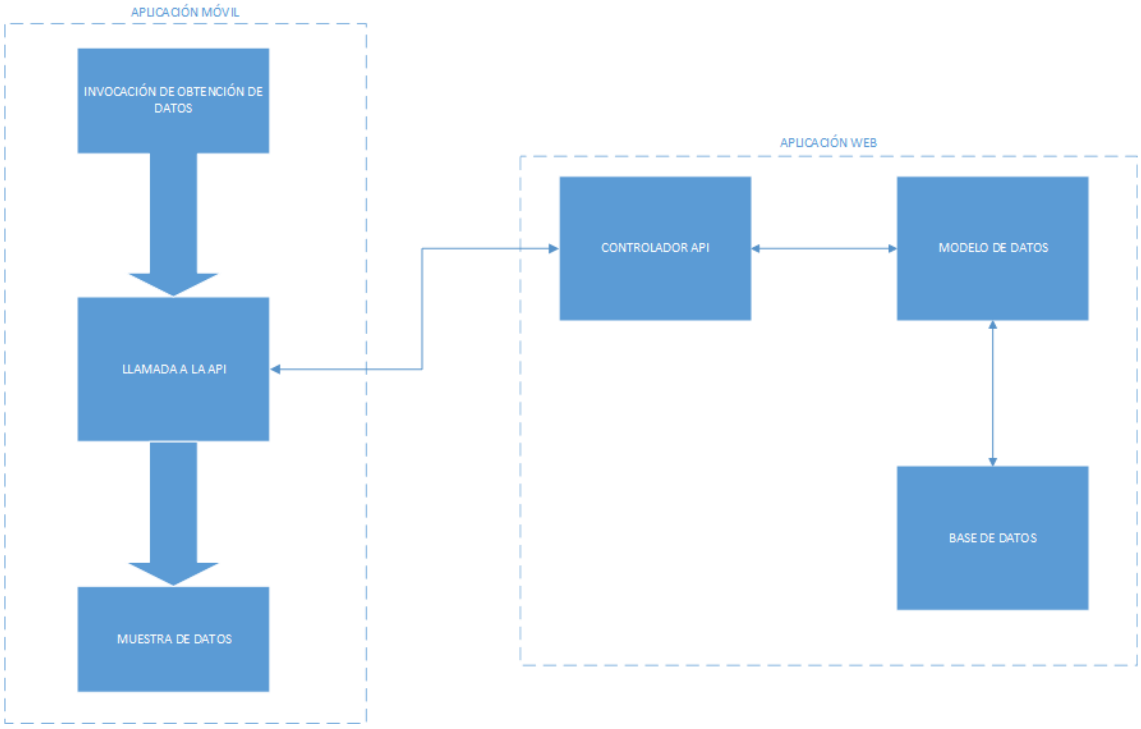


Figure 4.1: Diagrama de bloques

Por tanto, el sistema debe ser capaz de funcionar sin conexión a Internet por medio de otros sistemas de comunicación, como podría ser a través de emisoras de radio. Por ello, el sistema debe estar preparado para interactuar con dichos sistemas.

4.2. Tecnologías

Este proyecto, no disponía de ningún código fuente previo con el que continuar, por tanto, hemos tenido la oportunidad de poder realizar el desarrollo con las tecnologías que se han deseado.

El sistema, cuenta con tecnologías que han sido estudiadas y aprendidas durante el grado, pero también se han utilizado otras que no habían sido impartidas, bien en su totalidad o parcialmente.

Cabe destacar que todas ellas han sido pensadas conforme a los requisitos y restricciones anteriormente expuestos. También, se ha tenido en cuenta el factor de software libre y de capacidad (económica y de recursos) de los usuarios finales.

4.2.1. Git y GitHub

Como control de versiones, se ha utilizado Git y, aunque el desarrollo no ha sido en grupo, ha resultado conveniente utilizarlo, para evitar problemas de código.

Además, el código se encuentra almacenado en GitHub, lo que nos permite tener el código público para cualquiera que lo desee ver y hacer uso de sus herramientas.

La aplicación web se encuentra en el repositorio https://github.com/JIraizoz/GEPAME-web y la aplicación móvil en https://github.com/JIraizoz/GEPAME-movil.

4.2.2. ASP.NET Core

Para la aplicación web del sistema, tal y como se especifica en las restricciones del sistema, se ha utilizado el framework .NET Core en su versión 2.0.

Esta plataforma de Microsoft es una implementación reducida de .NET Standard modular, a través de paquetes de ensamblado administrados a través de NuGet; multiplataforma, ya que permite implementar las características que la aplicación necesite y es compatible con los principales sistemas operativos; y de código abierto, administrado por .NET Foundation y disponible en GitHub.

Las principales razones por las que se ha elegido, han sido la sencillez y facilidad que ofrece a la hora de trabajar y por el conocimiento del lenguaje, que era mayor que otros que se valoraron, como Java y Python.

Además, .NET Core, nos permite funcionar sobre casi cualquier servidor, bien sea Linux, Mac OS o Windows, con recursos mínimos, lo cual es idóneo para este sistema.

También, cabe destacar la oportunidad que se nos ofrece al poder desarrollar todo el sistema en un mismo lenguaje, tanto la aplicación web con ASP como la aplicación móvil con Xamarin. No obstante, no se optó por esa idea por una serie de motivos que se darán más adelante.

4.2.3. Android

La aplicación móvil, está desarrollada en Android con código nativo.

En un primer instante, se optó por desarrollar la aplicación móvil con Xamarin, una serie de herramientas multiplataforma que permite desarrollar aplicaciones para iOS, Android y UPW (Plataforma Universal de Windows) utilizando .NET, pero finalmente se desechó debido a la falta de conocimientos sobre la plataforma a la hora de implementar ciertas herramientas.

Por tanto, se decidió desarrollar la aplicación exclusivamente en Android, ya que el dominio era mayor, gracias a una aplicación desarrollada el curso anterior, y ofrecía más herramientas y el desarrollo era más sencillo. La versión mínima de la API es la 23 (Android 6.0).

4.2.4. HTML, CSS, JavaScript y Razor

Al tratarse de una aplicación web, y utilizar el modelo vista-controlador, el HTML es indispensable, al igual que el CSS y JavaScript.

Además, como se trata de una aplicación desarrollada en ASP.NET, se utiliza Razor para insertar código basado en servidor dentro de las vistas.

También, JavaScript ha sido utilizado para la implementación de la funcionalidad de mapas de Google API.

4.2.5. Entity Framework Core

Entity Framework (EF) Core es una versión ligera, extensible y multiplataforma de la popular tecnología de acceso a datos Entity Framework.

EF Core puede servir como asignador relacional de objetos, lo que permite a los desarrolladores de .NET trabajar con una base de datos mediante objetos .NET y eliminar la mayoría del código de acceso a los datos que normalmente deben escribir.

EF Core es compatible con muchos motores de base de datos, entre los que se destacan:

- SQL Server
- SQLite
- PostgreSQL
- MySQL
- MariaDB
- Firebird
- Oracle
- Access

Con EF Core, el acceso a datos se realiza mediante un modelo. Un modelo se compone de clases de entidad y un contexto derivado que representa una sesión con la base de datos, lo que permite consultar y guardar los datos.

Puede generar un modelo a partir de una base de datos existente, codificar manualmente un modelo para que coincida con la base de datos o usar migraciones de EF para crear una base de datos a partir del modelo.

Las instancias de las clases de entidad se recuperan de la base de datos mediante Language Integrated Query (LINQ), y los datos se crean, se eliminan y se modifican en la base de datos mediante instancias de las clases de entidad.

4.2.6. Swagger

Para la parte de la API REST implementada en la aplicación web, se ha utilizado el framework Swagger, que ofrece una plataforma en la que diseñar, documentar y probar cualquier API.

Además, ofrece al usuario un archivo de configuración en JSON para implementar de manera sencilla la API referenciada en un sistema.

Swagger, también nos ofrece la posibilidad de crear una API RESTful basada en la especificación estándar OpenAPI (OAS).

4.2.7. Auth0

Auth0 es una empresa que ofrece un servicio de autenticación y autorización de usuarios. Está pensado para ofrecer a los desarrolladores y compañías las herramientas necesarias para asegurar sus aplicaciones sin ser expertos en seguridad. Permite conectar cualquier aplicación, independientemente de su lenguaje, y definir la manera en la que los usuarios se conectan.

El servicio ofrece entre otras características:

- Inicio de sesión mediante usuario y contraseña o mediante redes sociales
- Seguridad de APIs
- Inicio de sesión único en varias aplicaciones
- Gestión de usuarios y contraseñas

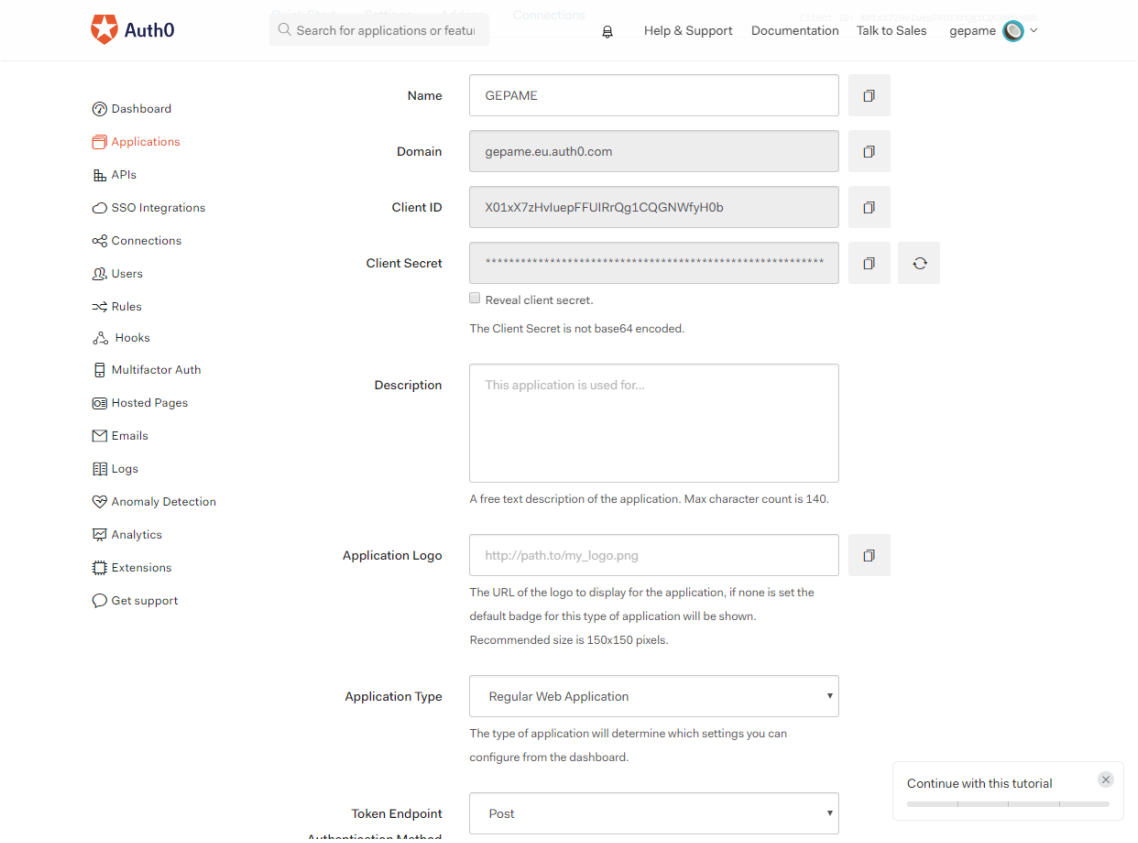


Figure 4.2: Panel de control de Auth0

A través de un panel de control, podemos gestionar las aplicaciones que queremos asegurar, así como las APIs, controlar los inicios de sesión, crear reglas, gestionar los usuarios, detectar anomalías, etc.

La herramienta, comparte con la aplicación los datos del usuario mediante tokens, permitiendo limitar el acceso a recursos en función del usuario u ofrecer diferentes servicios.

4.3. Hardware

El sistema, debe ser capaz de poder desplegarse en cualquier servidor, bien sea Linux o Windows, y debe ser capaz de ejecutarse sin problemas en un servidor con mínimos recursos, ya que, es posible que la organización que utilice el sistema no cuente con grandes recursos

Dado que se eligió .NET Core como *framework* de desarrollo, los sistemas sobre los que podemos desplegar la aplicación son varios, tal y como se muestran en el Cuadro 4.1.

SO	**Versión**	**Arquitectura**
Windows	≥7 SP1, 8.1	x64, x86
Windows 10	≥*Version* 1607	x64, x86
Windows Server	≥2008 R2 SP1	x64, x86
Mac OS X	≥10.12	x64
Red Hat Enterprise Linux	6	x64
Red Hat Enterprise Linux CentOS Oracle Linux	7	x64
Fedora	26, 27	x64
Debian	≥8.7, 9	x64
Ubuntu	14.04, 16.04, 17.10, 18.04	x64
Linux Mint	17, 18	x64
openSUSE	≥42.3	x64
SUSE Enterprise Linux (SLES)	≥12 SP2	x64

Table 4.1: Tabla de sistemas operativos soportados por .NET Core

Teniendo en cuenta que la aplicación debe funcionar con recursos limitados y la tecnología de desarrollo seleccionada, se ha decidido realizar la implementación en una Raspberry Pi 3 B con Raspbian 9 (stretch), ya que es un dispositivo de bajo coste (aproximadamente unos 40€) con capacidad para desplegar aplicaciones web.

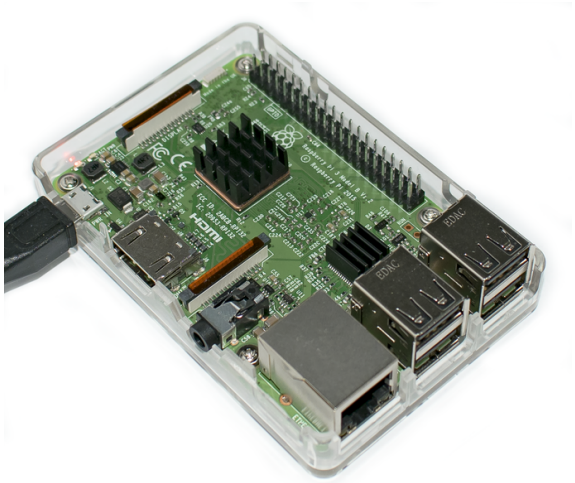


Figure 4.3: Raspberry Pi 3 B

Raspberry Pi es un ordenador de placa reducida de bajo costo que fue creado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de informática en las escuelas. De manera oficial, se ofrece un sistema operativo open-source, Raspbian, una versión adaptada de Debian, aunque se ofrecen múltiples sistemas distintos, incluyendo Windows 10 IoT.

En nuestro caso, la Raspberry Pi 3 B, dispone de las siguientes características técnicas:

- CPU: 1.2 GHz 64-bit quad-core ARM Cortex-A53
- SDRAM: 1 Gb
- Conectividad: 10/100 Mbit/s Ethernet, 802.11n WiFi, Bluetooth 4.1
- Consumo (máx): 1.34 A

En el caso de la aplicación móvil, se ha desarrollado en la API 26 de Android, con una versión mínima API 23 (Android 6.0).

En nuestro caso, se han utilizado dos dispositivos diferentes: un Blackview BV8000Pro (Android API 24) y un BQ Aquaris M4.5 (Android API 23).

	BQ Aquaris M4.5	**Blackview BV8000Pro**
CPU	MediaTek 4x Cortex-A53 a 1.0 GHz	MediaTek 8x Cortex-A53 a 2.3 GHz
RAM	2 Gb LPDDR3	6 Gb LPDDR4
Memoria	16 Gb	64 Gb
Pantalla	4.5”	5”
Batería	2470 mAh	4180 mAh
GPS	Sí	Sí, GPS + Glonass

Table 4.2: Características de los dispositivos utilizados

Chapter 5

Planificación

Para este proyecto, la planificación ha sido fundamental, no sólo por el hecho de que debía haber una organización para enfrentarse individualmente a este proyecto, sino porque el proyecto en el que se basaba tan sólo disponía de una breve descripción de ideas y no se contaba ni con especificaciones ni con código para desarrollar. Además, es necesario generar una documentación lo suficientemente adecuada para que otros pudiesen continuar con el trabajo que se ha empezado.

Por tanto, desde el primer momento, se decidió que, la planificación de este trabajo, debía estar pensada para la realización de un proyecto de software, no para un desarrollo.

Es por ello, por lo que se han especificado 7 tareas fundamentales:

1. Planificación
2. Especificaciones del sistema
3. Diseño del sistema
4. Desarrollo del sistema
5. Implementación del sistema
6. Pruebas
7. Generación de documentación

5.1. Planificación

En primer lugar, se hizo una planificación de todo el trabajo que debía realizarse, discernir qué fases de un proyecto de software eran fundamentales y cuáles podrían desarrollarse más tarde.

Además de eso, estaba la problemática de la gestión del tiempo, el cual, no ha sido bien distribuido por falta de experiencia en proyectos.

Dentro de la planificación, también se incluyó la investigación, necesaria para poder obtener funcionalidades del sistema y una panorámica de cómo se debían enfocar las siguientes partes y cuánta dedicación debía tener cada una.

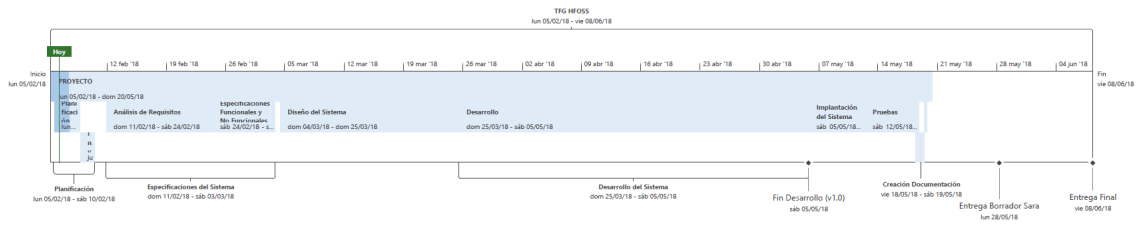


Figure 5.1: Planificación del TFG

A pesar de que se definieron bastantes características, las limitaciones de las tecnologías y el desconocimiento de las mismas, hicieron que el desarrollo no fuese de la manera que se esperaba, aunque se consiguió desarrollar un prototipo suficientemente funcional como para continuar el desarrollo.

5.2. Especificaciones del sistema

Durante esta fase, se realizó la captura de requisitos y la definición de las especificaciones de la aplicación. En esta fase, fue donde comenzaron a aparecer las primeras complicaciones.

Para la captura de requisitos, primero se hizo una selección y adecuación de las especificaciones del sistema anterior, ya que había ciertas funcionalidades que no interesaban, como la puntuación de conductores.

Con unos requisitos un tanto pobres, se decidió buscar ayuda en ciertas organizaciones, como la UME (Unidad Militar de Emergencias), aunque lamentablemente no se obtuvo respuesta alguna.

Dada la falta de éxito y de tiempo disponible, se tomó la decisión de crear los requisitos de manera autónoma, es decir, se buscó información en Internet sobre las funcionalidades que podría tener el sistema, proveniente de otros sistemas existentes en el mercado, y con la información obtenida y la propia intuición, se definió y especificó el sistema presentado.

La decisión de qué lenguaje de programación tomar y qué base de datos utilizar fueron las dos problemáticas más importantes.

Había que especificar un sistema capaz de ser desplegado sobre casi cualquier sistema operativo, y que pudiese utilizar una base de datos relacional independientemente del sistema gestor que se utilizase.

Dado que la planificación era muy ajustada, y algunas de las implementaciones sólo se conocían en ciertas tecnologías, se optó por utilizar .NET y SQL Server, aprovechando la suscripción *Dreamspark* que la universidad tiene con Microsoft, con el ánimo de conseguir desarrollar la mayor parte del sistema posible.

5.3. Diseño del sistema

Una vez definidos los requisitos, se procedió a realizar el diseño del sistema.

En primer lugar, se definieron los Casos de Uso, parte indispensable para el desarrollo, a partir de las funcionalidades del sistema.

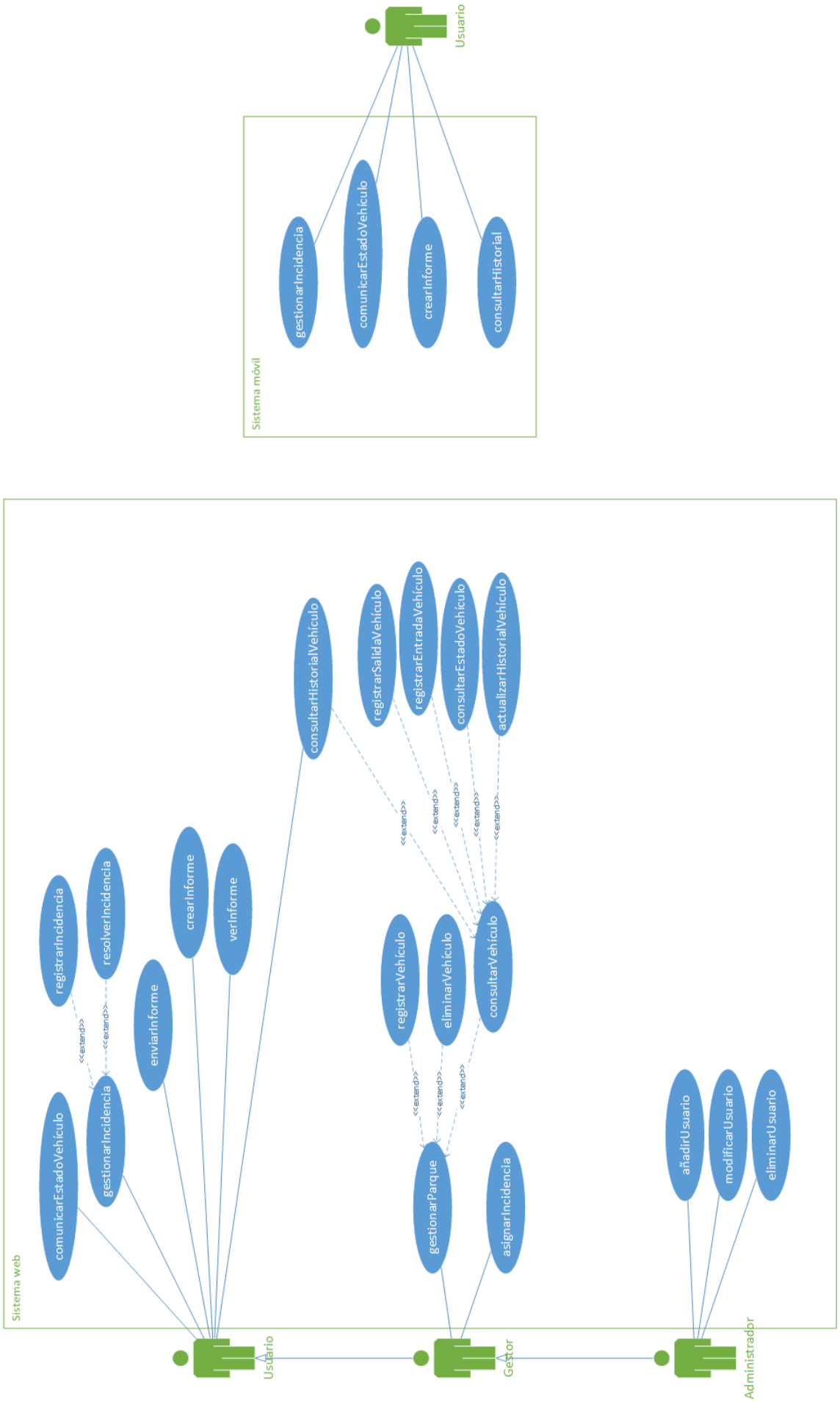


Figure 5.2: Diagrama de casos de uso de la aplicación web

Con las funcionalidades claras, se continuó con el diseño de la base de datos, una base de datos que debía ser sencilla, para evitar complicar la parte de desarrollo, y con la información necesaria, teniendo en cuenta que podría ser ampliable.

El sistema, sin embargo, no fue diseñado como suele ser habitual. Se optó por diseñar un boceto del sistema, estructurado en la arquitectura de tres capas, para acercar una idea de lo que se quería desarrollar.

Como se iba a utilizar el modelo vista-controlador y el modelado de datos sería definido a partir de la base de datos con el Entity Framework, se optó por no diseñar completamente el sistema, dando de esta manera mayor libertad y margen de maniobra a la hora de desarrollar.

5.4. Desarrollo del sistema

Esta fase, tampoco estuvo libre de complicaciones, es más, fue la parte en la que más problemas hubo.

La falta de conocimiento sobre algunos de los módulos, y la costumbre de trabajar con un framework de .NET completo, hizo que el desarrollo sufriese un gran retraso, el cual penalizó al desarrollo de la parte móvil.

No obstante, esa penalización no hubiese sido menor si se hubiese utilizado con otras tecnologías como Java o Python, al contrario más bien.

El desarrollo de la aplicación móvil, no solo perjudicada por el retraso acumulado, también se vio afectada por problemas con la tecnología acordada a utilizar.

En un principio, se decidió utilizar Xamarin para el desarrollo de la aplicación, con el fin de desarrollar ambas aplicaciones en un mismo lenguaje, para poder compartir código, y tener la posibilidad de crear una aplicación multiplataforma.

Tras una serie de problemas con ciertos paquetes, las conexiones con el API REST de la aplicación web y el retraso acumulado, se tomó la decisión de abandonar esa idea y centrarse en el desarrollo de un prototipo funcional en Android nativo, con Java y Android Studio.

Finalmente, se consiguió desarrollar una aplicación web y otra móvil con las funcionalidades básicas implementadas, preparada para seguir siendo desarrollada hasta una versión primera.

5.5. Implementación del sistema

En un primer instante, se pensó en la opción de desplegar la aplicación web en un servidor IIS (Internet Information Server), ya que se optó por desarrollar bajo tecnología de Microsoft, pero dado que una de las intenciones del sistema es que fuese un sistema multiplataforma, se decidió que la mejor opción era realizar la implementación en un servidor Linux.

Por tanto, y dado que se disponía de una Raspberry Pi, se procedió a desplegarla en este servidor. La Raspberry Pi cuenta con un sistema operativo Raspbian Stretch Lite, ampliado con SSH para administración remota y Apache como servidor web principal.

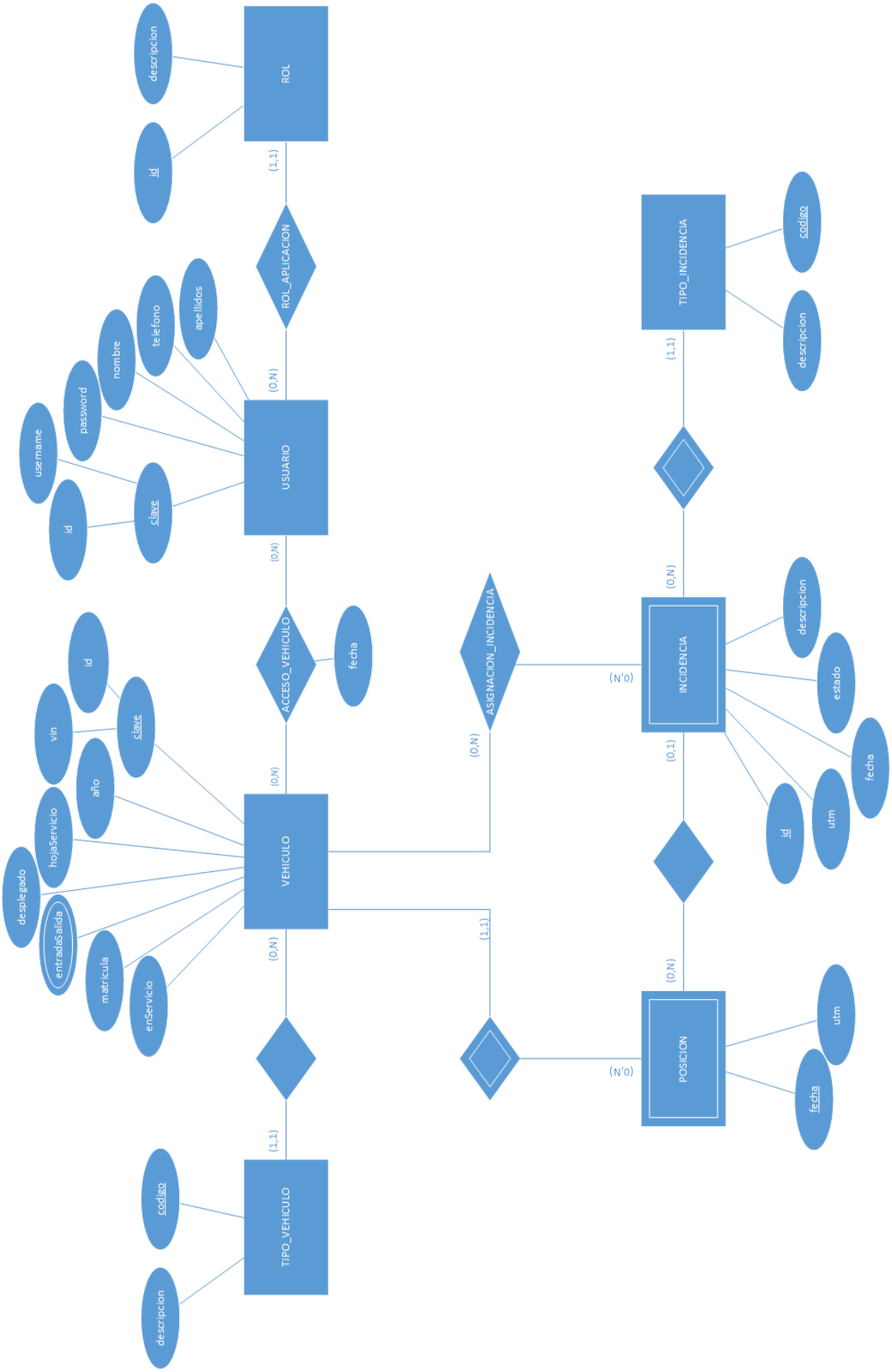


Figure 5.3: Diagrama Entidad-Relación de la base de datos

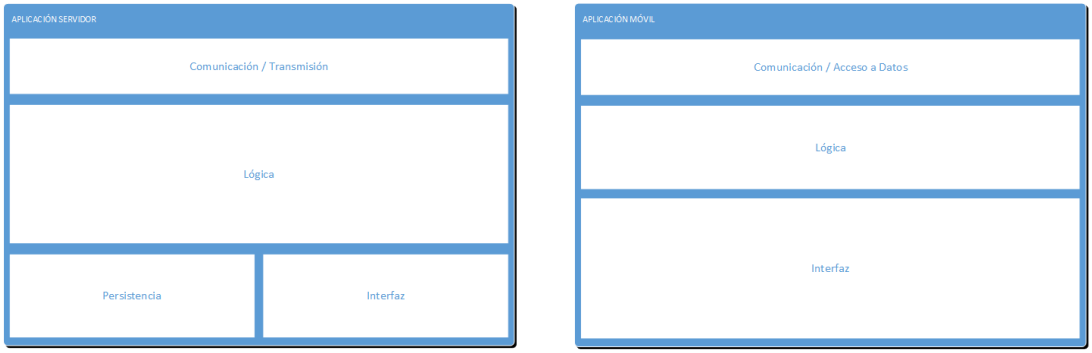


Figure 5.4: Diagrama conceptual del sistema

El principal inconveniente era desplegar una aplicación .NET sobre Linux, por lo que, consultando la documentación de .NET Core, se observó que, el propio framework, incluía un servidor para poder desplegar aplicaciones sobre otros sistemas que no fuesen Windows, Kestrel.

El único inconveniente era que tenían que coexistir dos servidores en un mismo dispositivo, Kestrel para la aplicación web y Apache para gestionar otras páginas alojadas en el mismo servidor.

No obstante, revisando la configuración, la solución pasaba por configurar el servidor Kestrel para que escuchase en local con un puerto designado (5000 en este caso) y que el servidor Apache actuase como proxy inverso. Puede verse el archivo de configuración en el apéndice A.

Para evitar problemas de acceso dinámico a los recursos de la aplicación, se creó y configuró un subdominio exclusivo para aplicación. De esta manera, se configuró un acceso a la aplicación sencillo y fiable.

En cuanto a la aplicación móvil, la generación del archivo de instalación APK y la instalación en los dispositivos de prueba se realizó de manera satisfactoria.

5.6. Pruebas

Las pruebas de la aplicación web, se realizaron en los siguientes exploradores web:

- Google Chrome (escritorio, tablet y móvil)
- Mozilla Firefox (escritorio y móvil)
- Microsoft Edge (escritorio y tablet)
- Internet Explorer (escritorio y tablet)

En todos los casos, la aplicación se comporta de manera esperada, aún siendo ejecutada sobre una Raspberry Pi 3. Cabe destacar, que en los dispositivos móviles, a pesar de que el diseño de las vistas es *responsive*, la información no adapta de manera adecuada.

Por último, la aplicación móvil fue probada en los dispositivos mencionados en el capítulo anterior, y en ambos, la aplicación se adapta correctamente y las funcionalidades implementadas funcionan de manera correcta.

5.7. Generación de documentación

Por último, y no menos importante, se encuentra la fase de documentación. Esta fase abarca tanto la creación de la documentación del proyecto como la de esta memoria y la presentación.

Dado el retraso acumulado de las otras fases, como el de la mala previsión temporal, se ha priorizado la realización de esta memoria, la cual contiene gran parte de la documentación del proyecto, y será utilizada para la realización de dicha documentación.

En su mayoría, el código se encuentra documentado, quedando pendiente la subida de la documentación a los correspondientes repositorios de GitHub.

Chapter 6

Resultados del desarrollo y Trabajo Futuro

6.1. Resultados del desarrollo

Finalmente, y tras superar todos los inconvenientes ocurridos durante todo el proyecto, el sistema cuenta con las siguientes funcionalidades desarrolladas:

6.1.1. Configuración del sistema

La aplicación web, a la cual se puede acceder desde gepame.jiraizoz.es y que está corriendo en una Raspbery Pi 3, se encuentra desplegada en un servidor web propio Kestrel, el cual viene predeterminado en la propia implementación de la aplicación, que permite:

- HTTPS
- Actualización opaca para habilitar WebSockets
- Sockets de Unix para alto rendimiento detrás de Nginx

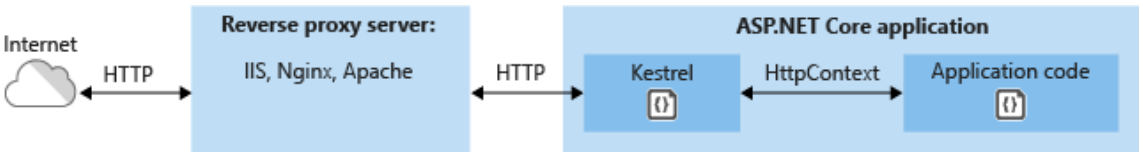


Figure 6.1: Diagrama de configuración de los servidores

A su vez, la Raspberry cuenta con un servidor web Apache, dedicado a la gestión del dominio en el que se aloja la aplicación, que actúa como proxy inverso para poder acceder a la aplicación.

En lo referente a la persistencia de los datos, utiliza un servidor SQL Server. Este servidor, se encuentra en un servicio virtualizado de Azure.

En ella, se almacena toda la información que la aplicación gestiona, desde las incidencias hasta los usuarios.

Sin embargo, la aplicación móvil, no accede directamente a dicha base de datos, sino que accede indirectamente a través de la API REST.

En la aplicación web, el sistema utiliza un módulo del Entity Framework Core, el cual nos ofrece un modelo de los datos en .NET y facilita la creación de los controladores API y el MVC.

6.1.2. Aplicación web

6.1.2.1. API REST + Swagger

REST nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que utilice HTTP.

Utilizando REST, obtenemos mayor facilidad de interacción con servicios desde plataformas sin mucha infraestructura.

Podemos hacer uso de los verbos HTTP (GET, POST, PUT y DELETE) para realizar operaciones CRUD (*Create*, *Read*, *Update*, *Delete*) sobre un conjunto de datos.

Para nuestro sistema, se creó una API REST preparada para la comunicación con la aplicación móvil, y pensada para interactuar con otros sistemas GEPAME.

Para crear la API, se definieron una serie de clases controladoras, que heredan de la clase Controller, con métodos para la definición de los verbos HTTP, acompañados de una serie de etiquetas de definición del recurso a servir. Puede verse un ejemplo de ello en los anexos.

Adicionalmente, se utilizó un módulo Swagger para ofrecer una interfaz gráfica de la API de consultas y pruebas, y para la generación de archivos de configuración automáticos para otros sistemas.

Los datos son enviados y recibidos en formato JSON, serializable a el objeto del modelo de datos solicitado.

Además, se ha configurado el framework Swagger para la realización de pruebas y la generación de documentos de configuración de conexión.

6.1.2.2. Gestión de vehículos

La aplicación es capaz de llevar un inventario de los vehículos disponibles, si se encuentran desplegados, y si se encuentran en servicio (si pueden ser utilizados)

Del mismo modo, se pueden dar de alta nuevos vehículos y editar o eliminar los ya existentes, además de los tipos de vehículos.

6.1.2.3. Gestión de incidencias

Desde el sistema, se pueden crear, editar y eliminar nuevas incidencias en un territorio, al igual que los tipos de ellas.

Las incidencias pueden ser consultadas en un listado o verse reflejadas en un mapa.

6.1.2.4. Gestión de usuarios

Con el fin de controlar el acceso a la página, el sistema puede crear, modificar y borrar usuarios del sistema.

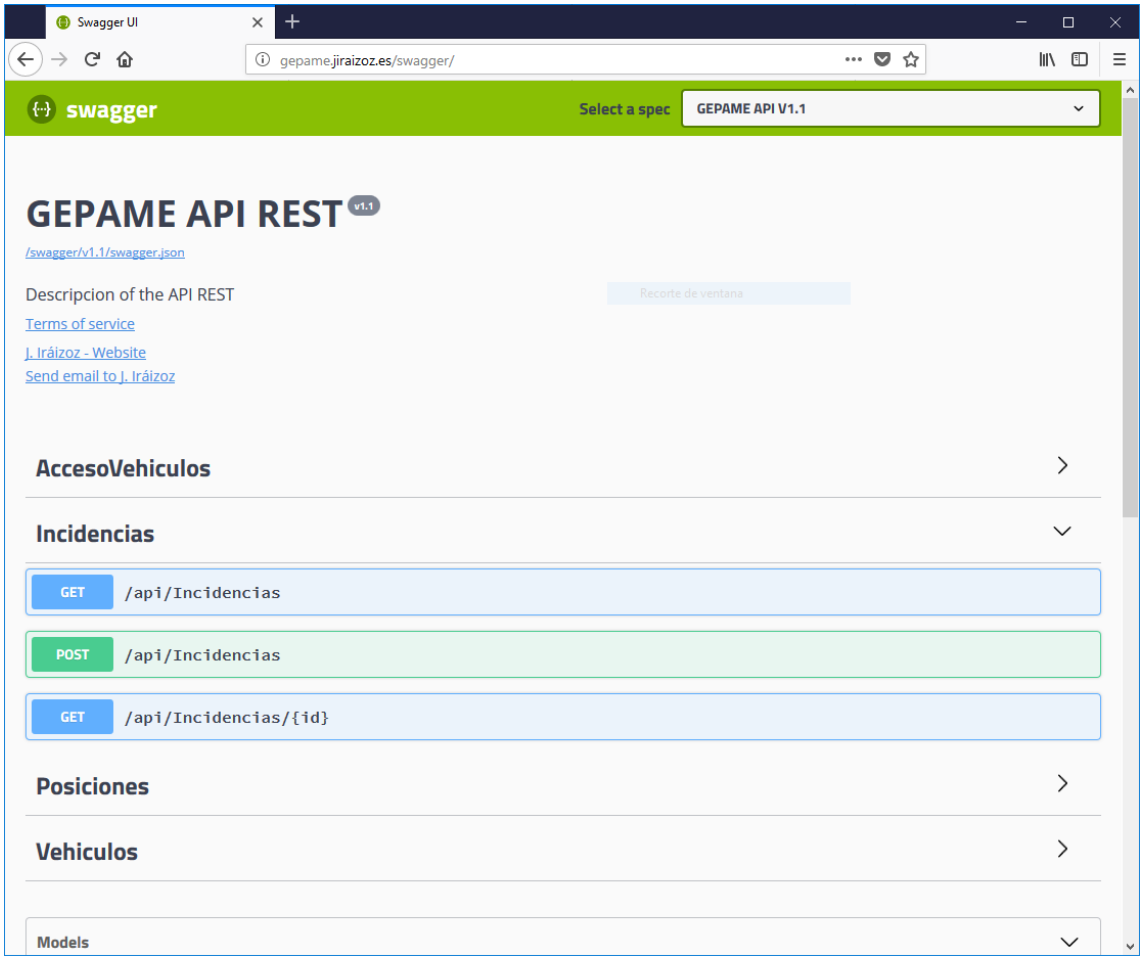


Figure 6.2: Ejemplo de consulta REST con Swagger

6.1.3. Aplicación móvil

6.1.3.1. Gestión de incidencias

Desde la aplicación móvil, la cual tiene menos funcionalidades que la web, se pueden consultar las incidencias existentes, tanto en un mapa como en un listado.

Si se selecciona la opción del listado, al acceder a una incidencia, esta te lleva al mapa mostrándote la incidencia seleccionada.

6.2. Trabajo futuro

A lo largo de este trabajo, como hemos visto, se han ido definiendo cada una de las diferentes partes de un proyecto de software, desde la captura de requisitos hasta la implementación del sistema en producción.

Recordemos, que el proyecto en el que se basa el nuestro, carecía de una descripción adecuada como para realizar un desarrollo. También, que el proyecto base, no disponía de código fuente con el que continuar trabajando, por tanto, se tuvieron que definir las funciones y especificaciones del

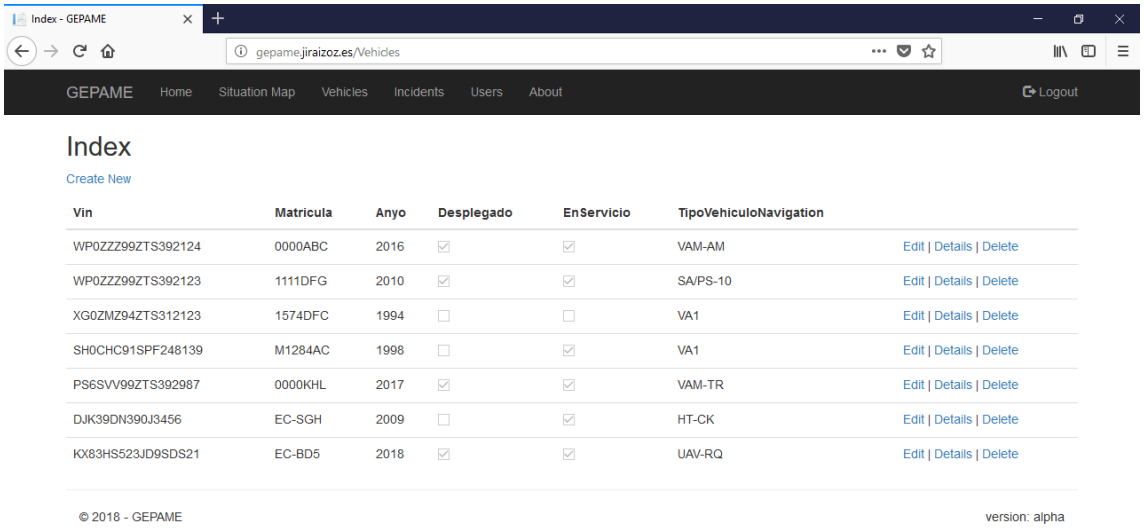


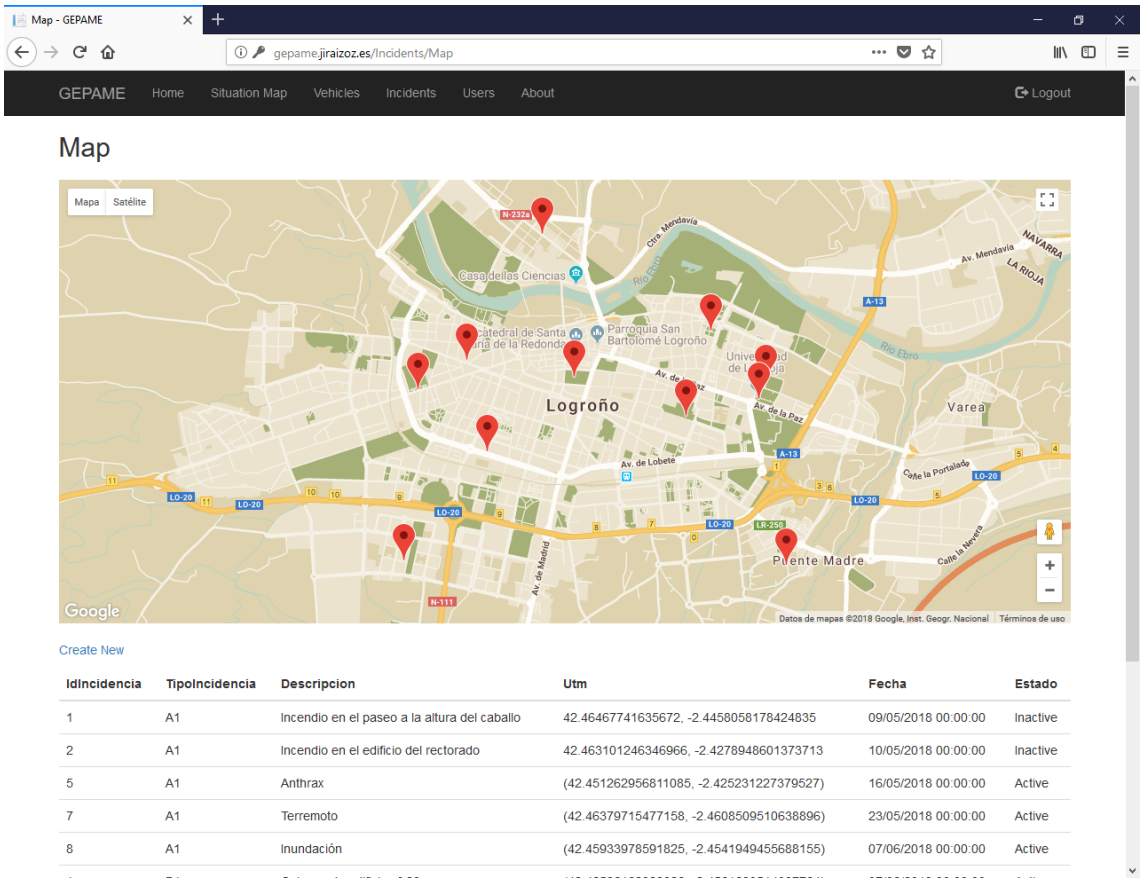
Figure 6.3: Ejemplo de listado de vehículos

sistema a desarrollar prácticamente en su totalidad.

Dado que, gran parte del tiempo y el esfuerzo fue dedicado a la especificación de requisitos y funcionalidades, existen algunas características del sistema que han quedado pendientes de desarrollar. Además, esto da la oportunidad de que otros desarrolladores se involucren con el proyecto y continúen con la tarea que hemos comenzado.

Destacar que, existen otras funcionalidades que no han sido definidas, que resultarían interesantes para el sistema, como podrían ser:

- Internacionalización de ambas aplicaciones
- Migración del código de la aplicación móvil de Android a Xamarin
- Encriptación y autenticación de las conexiones REST
- Descubrimiento de la ruta más rápida a una incidencia en función del resto de incidencias



IdIncidencia	TipoIncidencia	Descripcion	Utm	Fecha	Estado
1	A1	Incendio en el paseo a la altura del caballo	42.46467741635672, -2.4458058178424835	09/05/2018 00:00:00	Inactive
2	A1	Incendio en el edificio del rectorado	42.463101246346966, -2.427894860137371 3	10/05/2018 00:00:00	Inactive
5	A1	Anthrax	(42.451262956811085, -2.425231227379527)	16/05/2018 00:00:00	Active
7	A1	Terremoto	(42.46379715477158, -2.4608509510638896)	23/05/2018 00:00:00	Active
8	A1	Inundación	(42.45933978591825, -2.4541949455688155)	07/06/2018 00:00:00	Active

Figure 6.4: Ejemplo de incidencias en mapa

Chapter 7

Conclusión

La realización de este proyecto, representa una auténtica satisfacción a nivel personal. Es el resultado del esfuerzo, dedicación y superación durante todos los años de la carrera, y en especial de los últimos.

El hecho de finalizar esta etapa, ha sido motivación suficiente para superar todos los problemas que han ido surgiendo y continuar trabajando por conseguirlo. También, el papel de Sara como directora ha sido fundamental, y vuelvo a agradecerle el interés y la confianza depositada, así como el altruismo que me ha hecho ver en la informática. Pensar que algo que he desarrollado pueda servir para que otros aprendan, desarrollen o utilicen, te llena de energía para seguir trabajando y siendo quién eres.

Como se comentó en la introducción, este proyecto nace de un módulo de la aplicación Sahana EDEN, dedicado a la gestión de vehículos de emergencia. El módulo con el que se comenzó, tan sólo disponía de una descripción insuficiente como para comenzar un desarrollo, además de que no disponía código fuente, por lo que tuvo rehacerse el proyecto desde cero. La extracción de información y la investigación fueron claves para llegar a definir el sistema que se ha expuesto.

La ventaja de comenzar un proyecto nuevo estuvo, principalmente, en la libertad para escoger las diferentes tecnologías que se utilizan. No obstante, no está libre de inconvenientes, ya que, conseguir que la aplicación fuese multiplataforma y que pudiese ser ejecutada con mínimos recursos, suponía una fuerte investigación y un aprendizaje autónomo.

Finalmente, se ha conseguido crear un proyecto de software con suficiente información como para continuar desarrollando, teniendo en cuenta la naturaleza de HFOSS de este trabajo, partiendo de una idea sin información técnica y sin ayuda de ningún tipo, lo que llevó a realizar una fuerte investigación e impulsó el componente autodidacta presente (y necesario) en la informática, para acabar realizando el desarrollo y la implementación de un sistema funcional con el que empezar a trabajar.

Me hubiese gustado haber realizado este trabajo en grupo, como en un principio estaba previsto, se hubiese conseguido mucho más, pero por ciertas razones de entendimiento, no pudo ser así. Del mismo modo, hubiese sido gratificante y enriquecedor que alguna organización hubiese colaborado, en especial en la captura de requisitos, ya que, al no conocer el entorno real de este

sistema, desarrollar una aplicación con tan poca información resulta complicado, aunque también te hace ponerte en la piel del que acabará usándola y ver qué funcionalidades podrían ser útiles.

La aplicación, como se ha dicho anteriormente, tiene aún funcionalidades pendientes de implementar, pero espero que sirva como punto de partida para que otras personas sigan desarrollando con una base más sólida que de la que yo comencé. También, confío en que este sistema sirva para lo que fue diseñado, ayudar en la gestión de situaciones de emergencia y, por tanto, sea capaz de salvar vidas.

Chapter 8

Conclusion

The fulfillment of this project has been a real personal satisfaction. It's the result of my effort, dedication and self-improvement during all this degree years and, specially, the last of them.

The simple fact to end this step, has been enough motivation to overcome all problems and to carry on working to reach it. Also, Sara's role as my tutor has been essential, and I'll thank her again for her interest and trust she put on me, as the altruism she shown me on regarding computer science. Developing something that could be useful for others to learn, develop or just use it, it fills you of energy to continue working and being who you are.

As we introduced in this document, this project come from a Sahana EDEN module, dedicated to fleet emergency management. The module, just had an inadequate description as starting, also the missing source code, so we had to start a project from the beginning. The extraction of information and the researching were key to reach the exposed system.

The advantage of starting a new project, were on the freedom for choose the technologies we desired. Nevertheless, it's not free of issues, because make a multi-platform system and able to execute it with minimum resources, mean a strong investigation and self-learning.

Finally, we've got a software project with enough information for continue the developing, keeping in mind the HFOSS nature of this project, starting with an idea without technical information and without any help, what driven to make a strong investigation and awake a self-taught present (and necessary) in computer science, for reach a functional system.

I would have liked to do this project in group, as it was planned in the beginning, we could have achieved more, but due to some misunderstanding problems, it couldn't be. Also, it would have been gratifying and enriching if some organization could have help me, especially during requirement, now, when you don't know the environment of the system, it's complicated to develop an application with such a little information. Nevertheless, it drives you to put yourself on someone else's shoes, in this case, the final user's shoes, and at the end it is showing what and why it could be useful.

As I mentioned before, the application has some functionalities pending of being implemented, but I hope that this project will be a beginning for thous who will continue developing this with

more information that I had. And I also trust that this system achieve it propose, helping on the emergency management and, thus, save lives.

Bibliografía

[1] Sahana Foundation. *Sahana Foundation - Open Source Disaster Management Solutions*. Recuperado de https://sahanafoundation.org/

[2] qgibson662. *BluePrint/Vehicle/ProjMgt - SahanaEden*. Recuperado de http://eden.sahanafoundation.org/wiki/BluePrint/Vehicle/ProjMgt

[3] Francisco J. García Izquierdo. Universidad de la Rioja. Departamento de Matemáticas y Computación. *REST - Web Services*

[4] Microsoft. *Guía de .NET Core*. https://docs.microsoft.com/es-es/dotnet/core/index

[5] Microsoft. *Entity Framework Core*. https://docs.microsoft.com/es-es/ef/core/

[6] Microsoft. Sintaxis de Razor. Recuperado de https://docs.microsoft.com/es-es/aspnet/core/mvc/views/razor?view=aspnetcore-2.0

[7] R. Elmarsri, S. B. Navathe. Fundamentos de Sistemas de Bases de Datos. Ed. Pearson

[8] Auth0 Inc. Recuperado de https://auth0.com/

[9] Kerkton Security Technologies LLC. Security Computer-Aided Dispatch Software. Recuperado de https://www.kerkton.com/

[10] Hexagon AB. Intergraph Computer-Aided Dispatch. Recuperado de https://www.hexagonsafetyinfrastructure.com/products/command-control-and-communications/intergraph-computer-aided-dispatch

[11] Android. API Reference. https://developer.android.com/reference/?hl=es-419

[12] Google. Google Developers. Recuperado de https://developers.google.com/

[13] Smartbear. Swagger. Recuperado de https://swagger.io/

[14] Raspberry Pi. Recuperado de https://www.raspberrypi.org

[15] Wiki ELP UCM. Raspberry Pi. Recuperado de http://wikis.fdi.ucm.es/ELP/Raspberry_Pi

[16] I. Hernández, J. Iráizoz, S. Mauleón, D. Ortega. Mobile Ad Hoc Network (MANET). Recuperado de https://jiraizoz.es/MANET.pdf

[17] Apache Software Foundation. Documentación del Servidor de HTTP Apache. Recuperado de https://httpd.apache.org/docs/current/

Apéndice A

Configuración del Proxy Inverso

```
<VirtualHost *:80>
        DocumentRoot [RUTA_DIRECTORIO]
        ProxyPreserveHost On ProxyPass "/" "http://127.0.0.1:5000/"
        ProxyPassReverse "/" "http://127.0.0.1:5000/"
        ServerName gepame.jiraizoz.es
        ServerAlias gepame.jiraizoz.es
        ErrorLog [RUTA_DIRECTORIO]gepame-error.log
        CustomLog [RUTA_DIRECTORIO]gepame-access.log common
</VirtualHost>
```

Apéndice B

Ejemplo de Controlador de API

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using GEPAME.Models;

namespace GEPAME.Controllers.api
{
    [Produces("application/json")]
    [Route("api/Incidencias")]
    public class IncidenciasController : Controller
    {
        private readonly GEPAMEContext _context;

        public IncidenciasController(GEPAMEContext context)
        {
            _context = context;
        }

          // GET: api/Incidencias
          /// <summary>
          /// Gets a list of active Incidencias
          /// </summary>
          /// <returns>IEnumerable of Incidencia</returns>
          [HttpGet]
          public IEnumerable<Incidencia> GetIncidencia()
          {
              return from inc in _context.Incidencia
                     where inc.Estado
                     select inc;
          }

          // GET: api/Incidencias/5
```

```
[HttpGet("{id}")]
public async Task<IActionResult> GetIncidencia([
   FromRoute] string id)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var incidencia = await _context.Incidencia.
       SingleOrDefaultAsync(m => m.TipoIncidencia == id)
       ;

    if (incidencia == null)
    {
        return NotFound();
    }

    return Ok(incidencia);
}

// POST: api/Incidencias
[HttpPost]
public async Task<IActionResult> PostIncidencia([
   FromBody] Incidencia incidencia)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    _context.Incidencia.Add(incidencia);
    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateException)
    {
        if (IncidenciaExists(incidencia.TipoIncidencia))
        {
            return new StatusCodeResult(StatusCodes.
               Status409Conflict);
        }
        else
        {
            throw;
        }
    }
```

```
        return CreatedAtAction("GetIncidencia", new { id =
            incidencia.TipoIncidencia }, incidencia);
    }

    private bool IncidenciaExists(string id)
    {
        return _context.Incidencia.Any(e => e.TipoIncidencia
            == id);
    }
```